Automatisierte Suche nach Entwurfsmustern in Quelltext

Detlef Streitferdt, Ilka Philippow, Christian Heller

Fachgebiet Softwaresysteme / Prozessinformatik
Technische Universität Ilmenau
Postfach 100565
98684 Ilmenau
detlef.streitferdt@tu-ilmenau.de
ilka.philippow@tu-ilmenau.de
christian.heller@tu-ilmenau.de

Abstract: Die Wartung von Software ist eine zeitaufwendige Aktivität innerhalb des Softwarelebenszyklus, die ein gutes Verständnis des zu wartenden Systems voraussetzt. Schlecht oder gar nicht dokumentierte *Legacy*-Systeme erschweren jedoch das Verständnis oder verhindern es gar gänzlich. Um dennoch unbekannte objektorientierte System verstehen zu können, wird zunächst deren Architektur durch Analyse des Quelltextes zurückgewonnen. Allerdings ist die entstehende Klassenstruktur oft noch zu komplex, um sich schnell einen Überblick zu verschaffen und damit das System verstehen zu können. An dieser Stelle kann das Wissen um Entwurfsmuster Entwicklern helfen ein System schneller und besser zu verstehen. Mit Hilfe der aus dem *Information Retrieval* stammenden Leistungsmerkmale *recall* und *precision* wurden existierende Ansätze der Mustersuche untersucht und bewertet. Basierend auf dieser Untersuchung wurden 23 neue Algorithmen entwickelt, zur Mustersuche in Quelltext für die in [Gamm 1995] beschriebenen Entwurfsmuster. Damit ist eine automatisierte Mustersuche möglich, die das Verständnis existierender Systeme vereinfacht und beschleunigt. In diesem Beitrag werden erste Ergebnisse des neuen Ansatzes zur Mustersuche präsentiert und die Implementierung als Plug-in für die Entwicklungsumgebung *Together* vorgestellt. Die Forschungsergebnisse sind im Rahmen des InPULSE Projektes [INPU 2004] entstanden, welches vom BMBF [BMBF 2004] finanziert wurde.

1 Einleitung

Innerhalb des Softwarelebenszyklus zielen die Wartungsaktivitäten auf die Verwaltung und Integration neuer oder veränderter Anforderungen ab. Zur Erfüllung dieser Aufgaben müssen Softwareentwickler ein System vollständig verstehen, obwohl die Dokumentation und/oder Entwurfsmodelle oftmals fehlen oder von schlechter Qualität sind. Meist steht lediglich der Quelltext des Systems zur Verfügung, als die einfachste und verlässlichste Form der Dokumentation.

Entwurfsmuster stellen vorgefertigte und bereits getestete Lösungen fundamentaler Ent-

wicklungsprobleme dar. Die Nutzung von Entwurfsmustern ist für neue und junge Entwickler von Vorteil, da ihnen dadurch das Wissen der erfahrenen Entwickler zugänglich gemacht wird. Die Kenntnis von Entwurfsmustern in existierenden Systemen zusammen mit der Zuordnung der Klassen im Quelltext zu den Entwurfsmusterstrukturen führt zu einem verbesserten Verständnis des Systems. Muster werden im Quelltext nicht explizit beschrieben, abgesehen von seltenen Kommentaren im Quelltext. Die Informationen über Entwurfsmuster in einem Softwaresystem stecken implizit in dessen Architektur und müssen in den meisten Fällen manuell wiedergewonnen werden.

In diesem Beitrag werden erste Ergebnisse eines neuen Ansatzes zur automatisierten Mustersuche in Quelltext vorgestellt sowie die Implementierung der Suchalgorithmen als Plug-in für die Entwicklungsumgebung *Together* präsentiert. Die Suchalgorithmen stellen eine Erweiterung existierender Ansätze dar und basieren auf der Suche nach minimalen Schlüsselstrukturen. Unser Ansatz umfasst die in [Gamm 1995] beschriebenen Entwurfsmuster, die den de-facto Standard für Entwurfsmuster darstellen. Der Vorteil unseres Ansatzes besteht in der vollständigen Abdeckung der Gamma-Muster durch verbesserte Suchalgorithmen. Die Integration dieser Suchalgorithmen in die Entwicklungsumgebung *Together* ermöglicht das verbesserte und schnellere Verständnis von Softwaresystemen anhand von existierendem und schlecht bzw. gar nicht dokumentierten Quelltext.

2 Stand der Technik

Derzeit existierende Methoden der automatisierten Mustersuche wurden untersucht und nach den erzielten Suchergebnissen bewertet. Abhängig von den jeweils gefundenen und tatsächlich vorhandenen Mustern wurden die Suchergebnisse quantifiziert. Im Rahmen einer Mustersuche werden drei Fälle unterschieden:

- *True positive*, falls ein Muster erkannt wurde und tatsächlich im Quelltext vorhanden ist. Dieser Fall wird angestrebt.
- *False positive*, falls ein Muster erkannt wurde ohne tatsächlich implementiert worden zu sein. Dieser Fall sollte vermieden werden.
- *False negative*, falls ein tatsächlich implementiertes Muster nicht erkannt wurde. Dieser Fall sollte ebenfalls vermieden werden.

Basierend auf den jeweiligen Ergebnissen können Metriken für die Bewertung der Suchansätze und Werkzeuge abgeleitet werden. Mit Hilfe der Werte *recall* und *precision*, die aus dem Bereich des *Information Retrieval* [Salt 1983] stammen, wurden die Ansätze bewertet.

Dabei bezeichnet *recall* die Anzahl aller implementierten Muster eines Softwaresystems geteilt durch die Anzahl der gefundenen Muster. Ein *recall*-Wert von 100% steht für ein Suchergebnis bei zumindest alle implementierten Muster gefunden wurden. Es können auch mehr Muster erkannt worden sein. Die tatsächlich implementierten Muster wurden aber auf jeden Fall erkannt – dies entspricht dem 2. Fall, *false positive* wurde damit vermieden.

Die *precision* bezeichnet das Verhältnis zwischen erkannten und tatsächlich implementierten Mustern (*true positive*) geteilt durch die Anzahl der erkannten Muster, also der Summe aus den *true positive* und *false positive* Fällen. Eine *precision* von 50% sagt aus, dass die Hälfte der erkannten Muster nicht in dem untersuchten Softwaresystem implementiert waren. Beide Werte müssen für eine Bewertung von Werkzeugen mit in Betracht gezogen werden, wobei eine *precision* von 100% die *false positive* Fälle nicht ausschließt.

Name	Werkzeug	abgedeckte Muster	untersuchte Softwaresysteme	Recall	Precision
DP++ [Bans 1998] (Muster in C++)	ja	Kompositum, Dekorierer, Adapter, Fassade, Brücke, Fliegengewicht, Schablonenmethode, Zuständigkeitskette	Drawing Tool Kit (DTK) (44 Klassen)	keine Angabe	keine Angabe
KT [Brow 1996] (Muster in Smalltalk)	ja	Kompositum, Dekorierer, Zuständigkeitskette, Schablonenmethode, Strategie, Zustand, Befehl	System A (62 Klassen), System B (264 Klassen), System C (KT) (46 Klassen), System D (40 Klassen)	keine Angabe	keine Angabe
SPOOL [Kel2 1999] (Muster in C++)	ja	Schablonenmethode, Fabrikmethode, Brücke	System A (3103 Klassen), System B (1420 Klassen), ET++ (722 Klassen)	keine Angabe	keine Angabe
Pat [Krae 1996] (Muster in C++)	ja	Adapter, Brücke, Kompositum, Dekorierer, Proxy	NME (9 Klassen), LEDA (150 Klassen), zApp (240 Klassen), ACD (343 Klassen)	100%	37%
IDEA [Berg 2000] (Muster in UML)	ja	Schablonenmethode, Proxy, Adapter, Brücke, Kompositum, Dekorierer, Fabrikmethode, Abstrakte Fabrik, Iterator, Beobachter, Prototyp	keine Angabe	keine Angabe	keine Angabe
Mehrstufiger Suchprozeß [Anto 1998] (Muster in C++ oder OMT)	ja	Adapter, Brücke, Proxy, Kompositum, Dekorierer	LEDA, libg++, galib, groff, mec, socket (keine Angabe zur Anzahl der Klassen)	100%	35,00%
Flexible Musterdefinition und Fuzzy Logik [Nier 2001] (Muster in Java)	nein	alle			
Pattern Wizard [Kim 2000] (Muster in C++)	ja	alle	System 1, System 2, System 3 (keine Angabe zur Anzahl der Klassen)	keine Angabe	44%
BACKDOOR [Shull 1996]	nein	alle			

Tabelle 1: Untersuchungsergebnis existierender Ansätze zur Mustersuche

Das Ergebnis unserer Untersuchungen ist in Tabelle 1 dargestellt. Problematisch sind die jeweils angegebenen *precision* und *recall* Werte. Meist schwierig nachzuvollziehen, hängen alle Werte sehr stark von den untersuchten Softwaresystemen ab. Daher wurde parallel zu den Arbeiten an den Suchalgorithmen mit der Entwicklung eines Referenzsystems zur Mustersuche begonnen, welches den objektiven Vergleich von Suchalgorithmen für Entwurfsmuster ermöglicht.

Neben dem manuellen Ansatz von [Shull 1996] erfordert die Suche durch Metriken von [Kim 2000] und der *Fuzzy* Logik Ansatz von [Nier 2001] einen Mehraufwand für die Vorverarbeitung des Quelltextes, der sich bei allen anderen Verfahren erübrigt, da hier direkt der Quelltext untersucht werden kann. Wird von Quelltext ausgegangen und eine automatisierte Suche angestrebt, kommt damit nur die Suche nach Strukturen in Betracht. Im nächsten Kapitel wird die neu entwickelte Erweiterung der Mustersuche mit Hilfe von Strukturvergleichen vorgestellt.

3 Suche nach minimalen Schlüsselstrukturen

Die Neuerung des vorgestellten Ansatzes sind die für alle 23 Gamma-Muster definierten negativen Suchkriterien. Die Struktur jedes Entwurfsmusters wird durch die erwarteten und verbotenen Elemente beschrieben sowie durch Elemente deren Vorhandensein den Suchalgorithmus nicht beeinflusst.

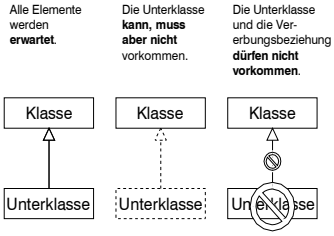

Abbildung 1: Suchkriterien

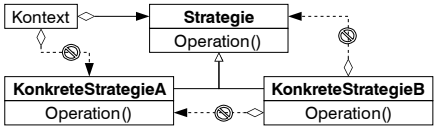

Abbildung 2: Suchkriterien, Strategie-Muster

In Abbildung 1 sind die grafischen Elemente zur Beschreibung der minimalen Schlüsselstruktur eines Musters dargestellt. Um ein Muster im Quelltext zu erkennen müssen aller erwarteten Elemente vorhanden sein, die verbotenen Elemente dürfen nicht vorhanden sein, während die Existenz der Elemente, wie sie im Mittelteil der Abbildung dargestellt sind, keinen Einfluss auf den Suchalgorithmus haben. Zu beachten ist jedoch, dass diese optionalen Elemente auch zwischen für die Musterstruktur relevanten Elementen liegen können.

Beispielhaft ist in Abbildung 2 die Suchstruktur für das Strategiemuster abgebildet. Die Aggregationsbeziehungen zwischen Kontext, KonkreteStrategieA und KonkreteStrategieB sind verboten, während die anderen Elemente vorkommen müssen, damit der entsprechende Suchalgorithmus ein Strategiemuster findet. In Quelltext 1 ist der den Suchkriterien entsprechende Pseudocode angegeben.

```
erstelle eine Menge der enthaltenen Bäume;
für jeden Baum i do
    ist Wurzelklasse (abstrakte Strategie) abstrakt?
    ja: für alle Unterklassen j (konkrete Strategien) do
        besitzt Klasse j dieselbe öffentliche Schnittstelle wie die Wurzelklasse?
        ja: besitzt Klasse j eine Referenz auf die Wurzelklasse?
            nein: besitzt Klasse j eine Referenz auf eine andere Unterklasse?
                nein: für alle Klassen k (Kontext) do
                    besitzt Klasse k eine Referenz auf die Wurzelklasse des
                    Baumes i?
                    ja: besitzt Klasse k weitere Referenzen auf die
                        Unterklassen des Baumes i?
                        nein: Muster gefunden

                od
    od
od
```

Quelltext 1: Pseudocode - Strategiemuster

3.1 Plug-in Architektur

Der Pseudocode aller 23 Suchalgorithmen wurde als Plug-in für die Entwicklungsumgebung *Together v6.0.1* in Java implementiert. Die derzeitige Implementierung greift über zwei APIs (*Application Programming Interfaces*) auf die von *Together* durch *Reverse-Engineering* gewonnen Modelldaten zu, wie in Abbildung 3 dargestellt. Diese Modelldaten liegen in Form von UML-Diagrammen (*Unified Modeling Language*) und dem entsprechenden Quelltext vor. Obwohl im Quelltext vorhanden, können nicht alle Informationen für die UML-Diagramme extrahiert werden. Zweideutigkeiten bei der Umsetzung von Quelltext in UML-Modelle führen zu weiteren Problemstellen im Datenmodell, was sich weiter auf die Suchalgorithmen auswirkt. In Tabelle 2 sind einige der Problemfälle der Umsetzung „Quelltext zu UML-Diagramm“ dargestellt.

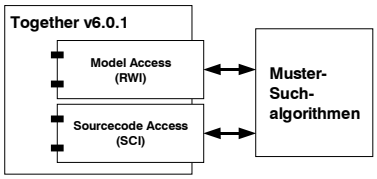

Abbildung 3: Plug-in Architektur

Quelltext	UML-Beziehung
B* ag;	A — ag — B A — 0..1 — ag — B
B kp;	A — kp — B
list<B> ag; vector<B> ag; deque<B> ag; OwnList ag;	A — 0..* — ag — B

Tabelle 2: Umsetzung Quelltext in UML

Insbesondere die 0..* Kardinalität der UML-Diagramme kann schlecht aus Quelltext extrahiert werden, betrachtet man die Vielzahl von Möglichkeiten diese Kardinalität in Verbindung mit Aggregation und Komposition in Java oder C++ zu implementieren. Darüber hinaus ist die von Gamma genutzte „*creates*“ Beziehung in der UML nicht definiert. Für die Implementierung der Suchalgorithmen wurde daher eine direkte Analyse des Quelltextes realisiert, welche die Erzeugung neuer Klassen, z.B. durch das C++-*new* Konstrukt, erkennt.

4 Ergebnisse der Mustersuche

Die Suchalgorithmen wurden an mehreren Systemen getestet. Bei dem ersten System, *Drawlet,* handelt es sich um ein Bildbearbeitungsprogramm mit 195 Klassen, welches von *Rolemodel Software* [RMS 2004] implementiert wurde. Als zweites System wurde das *Abstract Windowing Toolkit* (AWT) als Teil der *Java 2 Standard Edition* [J2SE 2004] mit insgesamt 354 Klassen untersucht. Schließlich wurde noch *Tomcat* als Teil des *Jakarta Open Source* Projektes [Tomc 2004] mit 1035 Klassen untersucht. Die Dokumentationen für AWT und *Tomcat* stammen von [Patt 2004], die des *Drawlet* Projektes von [Auer 1997]. Diese Dokumente waren die einzig verfügbaren Informationen, um verwendete Muster in den Projekten zu identifizieren und innerhalb der Architektur zu lokalisieren. Aufgrund der unzureichenden Genauigkeit der Dokumente werden in Tabelle 3

die Werte *precision* und *recall* für diese Systeme nicht angegeben. Zur Verbesserung der Suchergebnisse musste ein eigenes System entwickelt werden, welches alle der 23 Gamma-Muster enthält, genau in der von Gamma beschriebenen Form. Dieses Referenzsystem, als *Patterns* bezeichnet, wurde in Java implementiert und enthält 88 Klassen. Es enthält keinerlei Funktionalität, bildet jedoch exakt die Struktur der Muster ab und stellt das derzeit beste uns zur Verfügung stehende System für die Mustersuche dar. Für das *Patterns*-System wurden in Tabelle 3 die Werte für *precision* und *recall* angegeben.

Analysiertes System		**Drawlet**		**AWT**		**Tomcat**		**Patterns**			
Muster		enthalten	gefunden	enthalten	gefunden	enthalten	gefunden	enthalten	gefunden	precision	recall
Erzeugungsmuster	abstrakte Fabrik	n	3	j	10	n	4	1	1	100%	100%
	Erbauer	n	87	n	127	n	469	1	12	8%	100%
	Fabrikmethode	j	13	n	27	n	22	4	4	100%	100%
	Prototyp	j	n	n	11	n	2	2	2	100%	100%
	Singleton	n	n	j	8	n	6	1	1	100%	100%
Strukturmuster	Adapter	n	40	n	52	j	123	1	3	33%	100%
	Brücke	n	61	j	61	n	345	1	25	4%	100%
	Dekorierer	n	n	n	n	n	n	1	1	100%	100%
	Fassade	n	194	n	322	j	973	1	77	1%	100%
	Fliegengewicht	n	n	j	n	n	n	1	1	100%	100%
	Kompositum	j	n	j	n	n	n	1	1	100%	100%
	Proxy	n	9	n	14	j	73	1	3	33%	100%
Verhaltensmuster	Befehl	n	13	n	20	n	46	1	1	100%	100%
	Beobachter	j	n	j	n	j	n	1	1	100%	100%
	Besucher	n	1	n	6	n	4	1	1	100%	100%
	Interpreter	n	n	n	n	n	n	2	2	100%	100%
	Iterator	j	n	n	n	n	n	1	1	100%	100%
	Memento	j	n	n	n	n	3	1	1	100%	100%
	Schablonenmethode	j	1	j	22	n	17	1	1	100%	100%
	Strategie	j	4	n	4	j	12	1	17	6%	100%
	Vermittler	j	135	j	88	n	220	1	25	4%	100%
	Zustand	n	n	n	n	n	2	1	1	100%	100%
	Zuständigkeitskette	n	n	n	n	j	26	1	3	33%	100%
	In der Tabelle steht jeweils die Anzahl der enthaltenen bzw. gefunden Muster. j Das Muster wurde bei der Entwicklung des Systems verwendet, wir wissen jedoch nicht wie oft. n Es ist nicht bekannt, ob das Muster in dem System vorhanden ist oder nicht.										

Tabelle 3: Ergebnisse der Suchalgorithmen für Entwurfsmuster

Die Ergebnisse für *Drawlet*, *AWT* und *Tomcat* müssen zusammen mit den Werten für *precision* des *Patterns* Projektes gesehen werden. Bei den Einträgen mit einer hohen Zahl von gefundenen Mustern bei gleichzeitig niedriger Präzision sind weitere Forschung- und Entwicklungsarbeiten notwendig, da die Erkennungsgenauigkeit dieser Al-

gorithmen noch verbessert werden kann. Aus den Zeilen mit 100%-tiger *precision* ist jedoch ersichtlich, dass durchaus Muster Verwendung fanden und durch den hier vorgestellten Ansatz zur Mustersuche gefunden werden konnten. Die Ergebnisse des *Patterns*-Systems zeigen, dass 15 der 23 Suchalgorithmen korrekte Ergebnisse liefern. Die restlichen acht Algorithmen werden in Zukunft weiter analysiert, um deren Trefferquote zu erhöhen.

5 Zusammenfassung und Ausblick

In diesem Beitrag wurden erste Testergebnisse neu entwickelter Algorithmen für die Suche nach den 23 Gamma-Entwurfsmustern in Quelltext vorgestellt. Die Algorithmen wurden in Java als Plug-in für die Entwicklungsumgebung *Together v6.0.1* implementiert. Damit steht Entwicklern ein Werkzeug und Verfahren zur Verfügung, das die automatisierte Analyse eines lediglich im Quelltext vorliegenden Softwaresystems ermöglicht. Als Ergebnis dieser Analyse erlangt der Entwickler tiefergehende Kenntnisse der Zusammenhänge zwischen einzelnen Architekturteilen, durch die Verbindung dieser über Entwurfsmuster. Damit ist ein verbessertes und schnelleres Verständnis des Softwaresystems möglich.

Im Rahmen der Forschungsarbeiten hat sich herausgestellt, dass eine wichtige Voraussetzung für die erfolgreiche Suche nach Entwurfsmustern die Qualität des durch *Reverse Engineering* gewonnen UML-Modells eines zu untersuchenden Softwaresystems von großer Bedeutung ist. Dabei können Mehrdeutigkeiten zwischen Quelltext und UML-Modell zum Scheitern der Mustersuche führen. Zur Beseitigung dieser Probleme werden die Suchalgorithmen im Rahmen eines aktuellen Forschungsprojektes weiter verfeinert, so dass Mehrdeutigkeiten für alle entstehenden UML-Modellvarianten gesucht bzw. durch den Entwickler vor der Analyse der Quelltextes per Dialog eingestellt und damit eindeutig zugeordnet werden können. Parallel dazu wird bereits ein erweitertes Referenzsystem für die Mustersuche entwickelt, welches auch Implementierungsvarianten einzelner Muster enthält, die ebenfalls von den Suchalgorithmen gefunden werden sollen.

Literaturverzeichnis

[Anto 1998] G. Antoniol, R. Fiutem, L. Cristoforetti: Design pattern recovery in object-oriented software. In Proceeding of the 6th International Workshop on Program Comprehension (Ischia, Italy, June 1998), pp. 153-160, 1998.

[Auer 1997] Ken Auer: Fundamental Elements of an Extendible Java Framework. Rolemodel Software, www.rolemodelsoftware.com, 1997.

[Bans 1998] Jagdish Bansiya: Automatic Design-Pattern Identification. Dr. Dobbs Journal, www.ddj.com/articles/1998/9806/9806a/9806a.htm?topic=patterns, 1998.

[Berg 2000] Frederico Bergenti, Agostino Poggi: Improving UML designs using automatic design pattern detection. In Proceedings of the 12th International Conference on Software Engineering and Knowlege Engineering (SEKE 2000), pp. 336-343, Chicago, IL, 2000.

[BMBF 2004] BMBF: Bundesministerium für Bildung und Forschung. Web, www.bmbf.de, 2004.

[Brow 1996] Kyle Brown: Design reverse-engineering and automated design pattern detection in Smalltalk. Master's thesis, Department of Computer Engineering, North Carolina State University, hillside.net/patterns/papers, 1996.

[Gamm 1995] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[INPU 2004] InPULSE: Integrative Pattern- und UML orientierte Lern- und System-Entwicklungsumgebung. Web, www.inpulse-online.de, 2004.

[J2SE 2004] Sun.com: Java 2 Standard Edition. , java.sun.com/j2se/1.4.0, 2004.

[Kel2 1999] Rudolf K. Keller, Reinhard Schauer, Sebastian Robitaille, Patrick Page: Pattern-based reverse-engineering of design components. In Proceedings of the 21th Int. Conf. on Software Engineering, Los Angeles, USA, pages 226-235. IEEE Computer Society Press, 1999.

[Kim 2000] Hyoseob Kim, Cornelia Boldyreff: A method to recover design patterns using software product metrics. In Proceedings of the Sixth International Conference on Software Reuse (ICSR6), Vienna, Austria, 2000.

[Krae 1996] Christian Krämer, Lutz Prechelt: Design recovery by automated search for structural design patterns in object-oriented software. In Proceedings of the Working Conference on Reverse Engeneering, Monterey, CA, pp. 208-215, 1996.

[Nier 2001] Jörg Niere, Jörg P. Wadsack, Lothar Wendehals: Design pattern recovery based on source code analysis with fuzzy logic. Tech. Report tr-ri-01-222, University of Paderborn, Paderborn, Germany, www.upb.de/cs/ag-schaefer/Veroeffentlichungen/Quellen/Papers/2001/tr-ri-01-222.pdf, 2001.

[Patt 2004] Wiki-Server: PatternStories. Wikipedia, wiki.cs.uiuc.edu/PatternStories/DesignPatterns, 2004.

[RMS 2004] Rolemodel Software: Homepage. Web-Site, www.rolemodelsoftware.com/drawlets/index.php, 2004.

[Salt 1983] Salton G.: Introduction to Modern Information Retrieval. McGraw-Hill, New York, 1983.

[Shull 1996] Forrest Shull, Walcelio L. Melo, Victor R. Basili: An inductive method for discovering design patterns from object-oriented software systems. Technical Report UMIACS-TR-96-10, University of Maryland, 1991.

[Tomc 2004] Apache.com: Jakarta, Tomcat. Apache.org, jakarta.apache.org/tomcat, 2004.